CAPITOLO 21

Creazione di componenti dell'interfaccia utente

di David R. Chung

IN QUESTO CAPITOLO

L'Abstract Windowing Toolkit (AWT) di Java è composto da classi che incapsulano i controlli di base delle GUI. Poiché Java è una soluzione per qualsiasi piattaforma, l'AWT fornisce un'interfaccia che include le funzionalità comuni a tutte le interfacce. Qualsiasi interfaccia che si sviluppa dovrebbe essere per lo più uguale su qualsiasi piattaforma. Il Capitolo 13 contiene una descrizione generale dell'AWT.

L'AWT include molti controlli utili che possono essere sufficienti per le applicazioni o gli applet, a volte, tuttavia un'applicazione vera necessita di altro. Questo capitolo esamina due metodi per creare componenti personalizzati dell'interfaccia utente: l'estensione dei controlli e la combinazione di controlli.

Estensione dei controlli

In questo capitolo viene descritto come estendere la classe `TextField` per creare un controllo per le password. La nuova classe è un campo di testo che permette agli utenti di immettere una password; anziché visualizzare i caratteri digitati dall'utente, il controllo visualizza degli asterischi.

Per estendere il controllo si utilizza un meccanismo che prevede di derivare le classi. *Derivare* è un termine della programmazione orientata agli oggetti che significa modificare il modo in cui funziona una classe. Questo processo crea una nuova classe da quella vecchia e vi aggiunge nuove funzionalità.

La classe passField

La classe `passField` viene creata derivando la classe `TextField`. Il nuovo controllo permette all'utente di digitare una password. Per la classe viene fornita l'implementazione del metodo `processKeyEvent()`. Quando l'utente immette un carattere, questo metodo risponde e "consuma" l'evento. Quando l'utente preme un tasto, viene tenuta traccia del tasto premuto e nel controllo viene inserito un asterisco, in modo da tenere nascosta la password.

Questo controllo in realtà non verifica se la password è valida, ma tiene traccia dell'input dell'utente e nasconde i caratteri digitati. Più avanti viene spiegato come combinare questo controllo utile con altri metodi per creare un controllo per password.

Dati membro

La classe `passField` deve tenere traccia dei caratteri immessi dall'utente. Per farlo, necessita di un dato membro di tipo `String`:

```
import java.awt.*;
import java.awt.event.*;

class passField extends java.awt.TextField {
    String pass ;
```

Il costruttore della classe deve sapere quanti caratteri può contenere il controllo. Per trasmettere questa informazione, il costruttore della classe utilizza un parametro intero, `chars`. Poiché `passField` deriva da `TextField`, la prima riga del costruttore deve essere una chiamata al costruttore della superclasse.

Per capire come le classi derivino o estendano altre classi, è necessario conoscere il significato di alcuni termini della programmazione orientata agli oggetti. In questo esempio, `passField` *deriva da* `TextField`, *che è chiamata* superclasse, *o* classe genitore, *o* classe di base, *mentre* `passField` *è detta* sottoclasse, classe figlia *o* classe derivata.

La chiamata a `super(int)` in realtà richiama il costruttore della superclasse, `TextField(int)`. Il metodo `enableEvents()` indica che questa classe elabora eventi della tastiera. La riga successiva nel costruttore crea semplicemente la stringa:

```
public passField( int chars ) {
    super( chars ) ;
    enableEvents( AWTEvent.KEY_EVENT_MASK ) ;
    pass = new String() ;
}
```

Questo controllo viene derivato per nascondere l'input dell'utente. Per farlo, il controllo deve gestire l'input dell'utente nella classe derivata. Gli eventi vengono quindi *consumati*, in modo che la classe di base `TextField` non vi risponda. Per gestire l'input dell'utente, questo controllo ridefinisce il metodo `processKeyEvent()` della classe, che viene richiamato ogni volta che l'utente preme un tasto.

Il metodo `processKeyEvent()` ridefinito deve fare quanto segue.

- ✔ Aggiungere un asterisco al controllo.
- ✔ Memorizzare il valore reale immesso premendo il tasto.
- ✔ Posizionare il cursore alla fine della stringa.

Poiché questa classe in realtà non visualizza i valori immessi dall'utente, la chiamata a `getText()` restituisce una stringa di asterischi. Di seguito il metodo aggiunge un asterisco alla stringa e la reinserisce nel controllo. Il metodo `select()` viene utilizzato per posizionare il cursore alla fine della riga; poiché i due parametri sono uguali, in realtà non viene selezionato niente:

```
protected void processKeyEvent( KeyEvent e ) {
        switch ( e.getID() ) {
            case KeyEvent.KEY_PRESSED :
                String text = getText() ;
                setText( text + "*" ) ;
                select( text.length() + 1, text.length() + 1 ) ;
```

Il passo successivo consiste nel memorizzare il tasto premuto in una stringa. Poiché il parametro `key` è un `int`, è necessario convertirlo in un `char` e quindi utilizzare il metodo `String.valueOf()` per convertirlo in una stringa, che poi viene concatenata alla stringa esistente. Il metodo `processKeyEvent()` ha gestito completamente l'evento della tastiera. La chiamata a `consume()` impedisce a qualsiasi altro metodo di rispondere all'evento.

```
    pass = pass + String.valueOf( (char)key ) ;
    e.consume() ;
}
```

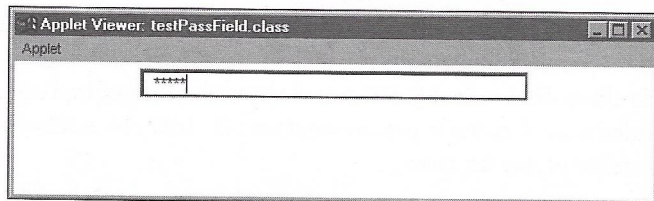
Il metodo `getString()` permette ai *contenitori* che utilizzano questo controllo di ottenere il valore immesso dall'utente:

```
    public String getString() {
        return pass ;
    }
}
```

Per verificare la classe `passField`, la si inserisce in un semplice applet, come indicato di seguito:

```
import java.awt.*;
public class testPassField extends java.applet.Applet {
    public void init() {
        add( new passField( 40 ) ) ;
    }
}
```

La Figura 21.1 mostra l'applet `testPassField`.

Figura 21.1
L'applet testPassField.

Se si esegue l'applet e si digitano delle lettere, si nota che vi sono dei comportamenti strani. Quando si preme il tasto [←], anziché cancellare un asterisco come ci si aspetta, il controllo ne aggiunge un altro. Il motivo è che, quando viene premuto un tasto qualsiasi, il metodo `processKeyEvent()` lo aggiunge alla stringa e visualizza un altro asterisco.

Per risolvere questo problema, è necessario gestire alcuni tasti in modo diverso. Se si richiama `consume()` solo per i tasti che si desidera gestire, il metodo `processKeyEvent()` della superclasse gestisce gli altri tasti. La chiamata a `consume()` può essere sostituita con il seguente codice:

```
if ( arg > 20 ) {
    e.consume();
}
```

Nonostante questa sia solo una soluzione parziale, è un inizio. Se il tasto premuto ha un valore ASCII inferiore a 20, vale a dire se è un carattere non stampabile, non viene consumato e viene gestito dal metodo `processKeyEvent()` della superclasse. Se il metodo `processKeyEvent()` consuma l'evento, questo non viene passato. In questo esempio, questa piccola modifica fa sì che il controllo accetti il tasto [←]. Per rendere davvero utile questo metodo, è necessario modificare anche il valore della stringa in modo che venga eliminato un asterisco.

Combinazione di controlli

Chi ha fatto parte qualche volta di un comitato, sa come sia difficile per un gruppo di persone lavorare insieme per raggiungere un obiettivo comune. Senza una leadership, sembra che tutti vadano per la loro strada. Senza comunicazioni ben coordinate, può avvenire una duplicazione degli sforzi. In modo simile, se si cerca di mettere insieme un applet di Java combinando diversi controlli dell'AWT, si può avere l'impressione di un grande comitato: molta attività, ma nessuna leadership e nessuna comunicazione.

Tuttavia, se si combinano i controlli in *controlli compositi*, questi agiranno come gli altri controlli dell'AWT. È possibile utilizzare questi nuovi controlli compositi in qualunque situazione in cui si utilizzano i normali controlli dell'AWT. Per mostrare i controlli compositi, nel seguito si spiega come creare il controllo di una finestra con un'immagine scorrevole. Questo controllo utilizza un'immagine e la rende scorrevole. Tutte le interazioni tra i componenti dell'AWT inclusi nel controllo vengono gestite internamente. Per utilizzare il controllo, è sufficiente crearlo e aggiungerlo al layout dell'applet.

Anche i pannelli sono componenti

Il vantaggio principale dell'utilizzo di `Panel` come classe di base dei componenti compositi è che `Panel` stessa è un componente. Di conseguenza è possibile utilizzare i componenti compositi come qualsiasi altro componente dell'AWT. È anche possibile combinarli per formare componenti compositi da altri componenti compositi e così via.

I nuovi componenti compositi possono essere aggiunti ai layout e possono generare eventi esistenti o crearne di nuovi. Sono componenti dell'interfaccia utente completi di tutte le funzionalità e possono essere utilizzati ogni volta che vengono utilizzati i componenti dell'AWT.

I controlli compositi sono più versatili se li si implementa con il gestore di layout appropriato. Poiché i controlli devono essere completi e autonomi, devono essere in grado di disporsi in modo appropriato, indipendentemente dalle loro dimensioni.

Utilizzo di pannelli per combinare elementi dell'interfaccia utente

La classe di base per tutti i controlli compositi è `Panel`, che permette di incorporare al suo interno altri componenti dell'AWT. Poiché questa classe deriva da `Container`, può contenere altri elementi dell'interfaccia utente. La classe `Panel` include inoltre funzioni per gestire i componenti incorporati.

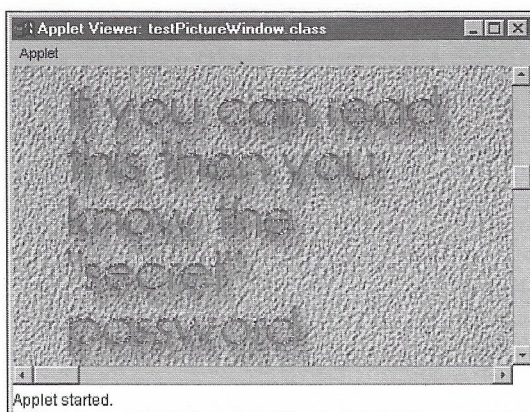
Alcune funzioni nella classe `Panel` possono richiamare riferimenti ai componenti incorporati. Queste funzioni permettono alla classe di richiamare metodi nei componenti incorporati in modo iterativo. Altre funzioni gestiscono i problemi relativi al layout.

Un esempio di finestra con un'immagine scorrevole

In questo esempio viene creata una finestra con un'immagine scorrevole. La classe `ScrollingPictureWindow` deriva da `Panel` e contiene tre oggetti membro: un oggetto `ImageCanvas` derivato da `Canvas` per contenere l'immagine e due barre di scorrimento.

Per rispondere alle barre di scorrimento, la classe `ScrollingPictureWindow` implementa `AdjustmentListener` e di conseguenza è un ascoltatore per le sue barre di scorrimento.

Questo controllo composito permette di visualizzare un'immagine. L'utente deve semplicemente passare un oggetto `Image` al controllo, il quale fa tutto il resto. Il controllo gestisce lo scorrimento e l'aggiornamento dell'immagine. Nella Figura 21.2 è mostrato l'applet con la finestra con l'immagine scorrevole.

Figura 21.2 *L'applet testPictureWindow.*

L'applet `testPictureWindow` *utilizza un oggetto* `ScrollingPictureWindow` *e crea il* `ScrollingPictureWindow` *esattamente nello stesso modo in cui si utilizzerebbe un controllo dell'AWT. Il codice sorgente della classe* `testPictureWindow` *si trova nel Listato 21.1, Questa classe e la classe* `ScrollingPictureWindow` *del Listato 21.2, presentato più avanti in questo capitolo, formano l'applet* `testPictureWindow`*. Entrambe le classi si trovano nel CD-ROM allegato al libro.*

Listato 21.1 *Il codice sorgente per la classe testPictureWindow.*

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class testPictureWindow extends Applet {

    ScrollingPictureWindow pictureWindow ;

public void init() {
    Image img = getImage( getCodeBase(), "picture.gif" ) ;
    pictureWindow = new ScrollingPictureWindow( img ) ;
    setLayout( new BorderLayout() );
    add( "Center", pictureWindow ) ;
    pictureWindow.setEnabled( true ) ;
    }
};
```

La classe ImageCanvas

La classe `ImageCanvas` deriva da `Canvas`, che è inclusa nell'AWT come classe generica per visualizzare e disegnare. Questa classe viene utilizzata per visualizzare l'immagine e contiene una variabile istanza:

```
Image canvasImg ;
```

Il costruttore di ImageCanvas utilizza come parametro un oggetto Image. Poiché i parametri oggetto vengono passati per riferimento, img diventa un riferimento locale all'oggetto Image nella classe:

```
public ImageCanvas( Image img ) {
    canvasImg = img ;
}
```

L'unico altro metodo incluso nella classe ImageCanvas è paint(), che disegna l'immagine. Prima di disegnarla, tuttavia, il controllo deve determinare se il suo genitore è attivato. Questa verifica permette di disattivare l'intero controllo.

Poiché l'immagine è scorrevole, la classe deve sapere dove disegnarla. La posizione dell'immagine dipende dalla posizione delle barre di scorrimento. In questo schema, l'oggetto ScrollingPictureWindow gestisce le comunicazioni tra gli oggetti membro. Per determinare dove disegnare l'immagine è necessario inviare una query a ScrollingPictureWindow:

```
public void paint(Graphics g) {
    if ( getParent().isEnabled() ) {
        g.drawImage( canvasImg,
          -1 * ((ScrollingPictureWindow)getParent()).imgX,
          -1 * ((ScrollingPictureWindow)getParent()).imgY,
          this ) ;
    }
}
```

Per ottenere le informazioni, si utilizza il metodo getParent(), che è un membro della classe Component che restituisce un riferimento all'oggetto Container che contiene il Component.

Quando si richiama getParent(), si ottiene un riferimento all'oggetto ScrollingPictureWindow. Poiché questo riferimento è il tipo Container, è necessario convertirlo in un riferimento a ScrollingPictureWindow. Ora è possibile accedere alle variabili istanza pubbliche nell'oggetto ScrollingPictureWindow. I membri imgX e imgY contengono le coordinate x e y del punto dell'Image che verrà visualizzato nell'angolo superiore sinistro della finestra. Se si desidera visualizzare il punto (10,5) nell'angolo superiore sinistro, si passa -10 e -5 a drawImage().

Variabili istanza

La classe ScrollingPictureWindow contiene diverse variabili istanza, che includono i controlli incorporati e le variabili di stato. I controlli incorporati sono memorizzati come segue:

```
public ImageCanvas imageCanvas ;
Scrollbar   vertBar ;
Scrollbar   horzBar ;
Image       image ;
```

L'ultima variabile istanza in questo elenco è un riferimento a un oggetto Image, che viene passato dal possessore dell'oggetto di classe. Le altre variabili istanza contengono tutte informazioni sullo stato del controllo. Le prime due contengono le dimensioni in pixel dell'intera immagine:

```
int imgWidth ;
int imgHeight ;
```

Le variabili istanza seguenti contengono la posizione corrente dell'immagine e riflettono anche la posizione corrente delle barre di scorrimento. Poiché le barre di scorrimento e l'immagine sono collegate, entrambe le classi utilizzano queste variabili. Le barre di scorrimento ne impostano i valori, mentre `ImageCanvas` utilizza questi valori per posizionare l'immagine:

```
int imgX ;
int imgY ;
```

L'ultima variabile è utilizzata dalle barre di scorrimento. Questo valore specifica di quanto si sposta la barra di scorrimento quando si preme [Pag↑] o [Pag↓]:

```
int block ;
```

Costruzione della classe

Il costruttore della classe esegue l'intera inizializzazione della classe e deve fare quanto segue.

- ✔ Inizializzare le variabili di stato.
- ✔ Determinare le dimensioni dell'immagine.
- ✔ Istanziare i controlli membro.
- ✔ Impostare il gestore di layout `GridBagLayout`.
- ✔ Impostare i valori per ogni controllo.
- ✔ Aggiungere la classe come ricevitore.

Inizializzazione delle variabili di stato

Il costruttore inizia impostando il riferimento locale a `Image` sull'argomento `Image`:

```
public ScrollingPictureWindow ( Image img ) {
    image = img ;
```

Il passaggio successivo è semplice. Si deve inizializzare `imgX` e `imgY` su zero. In questo modo si imposta la posizione iniziale dell'immagine e delle barre di scorrimento. Queste due variabili istanza contengono gli offset x e y in cui visualizzare l'immagine:

```
imgX = 0 ;
imgY = 0 ;
```

La classe `ImageCanvas` necessita di queste variabili per determinare il modo in cui posizionare l'immagine. Il metodo `ImageCanvas paint()` accede direttamente a queste variabili istanza e le utilizza nella chiamata a `drawImage()`.

Determinazione delle dimensioni dell'immagine

Il controllo composito deve ora conoscere le dimensioni dell'immagine. Quando si hanno queste informazioni, si sa che rimarranno costanti. Sfortunatamente, determinare le dimensioni dell'immagine non è così semplice come si potrebbe pensare.

La classe è stata progettata per utilizzare come parametro un oggetto Image, il che da agli utenti della classe una grande flessibilità per caricare l'immagine nel modo desiderato. L'immagine che si riceve potrebbe essere una di molte contenute in un array, potrebbe essere utilizzata da altri oggetti dell'applet e potrebbe anche essere stata appena caricata dall'applet. In quest'ultimo caso sorgono dei problemi.

Nel costruttore della classe, è possibile che il riferimento ricevuto punti a un'immagine non ancora completamente caricata; per ottenere le dimensioni, si effettua una chiamata a Image.getHeight(), ma se l'immagine non è caricata completamente, getHeight() restituisce -1. Per ottenere le dimensioni dell'immagine, è necessario eseguire un ciclo finché getHeight() restituisce un valore diverso da -1. Entrambi i seguenti cicli while hanno corpi vuoti:

```
while ((imgHeight = image.getHeight(this)) == -1 ) {
    // ciclo fino al caricamento dell'immagine
}

while ((imgWidth  = image.getWidth(this)) == -1 ) {
    // ciclo fino al caricamento dell'immagine
}
```

Istanziare i controlli membro

Ora è necessario creare gli oggetti membro incorporati. Il metodo ImageCanvas() utilizza come parametro l'oggetto Image. I costruttori delle barre di scorrimento utilizzano ognuno una costante che determina se la barra di scorrimento è verticale od orizzontale:

```
imageCanvas = new ImageCanvas( image ) ;

vertBar = new Scrollbar( Scrollbar.VERTICAL ) ;
horzBar = new Scrollbar( Scrollbar.HORIZONTAL ) ;
```

Impostazione del GridBagLayout

Per disporre i controlli incorporati si utilizza il gestore di layout GridBagLayout, che è il più versatile nell'AWT e che fornisce il controllo necessario per disporre i componenti.

Innanzitutto si crea un oggetto GridBagLayout, quindi si richiama setLayout() per fare in modo che diventi il gestore di layout corrente:

```
GridBagLayout gridbag = new GridBagLayout();
setLayout( gridbag ) ;
```

Impostazione dei valori per ogni controllo

La classe GridBagLayout utilizza la classe GridBagConstraints per specificare il modo in cui vengono disposti i controlli. Innanzitutto si crea un oggetto GridBagConstraints, quindi lo si utilizza per determinare la disposizione dei singoli componenti:

```
GridBagConstraints c = new GridBagConstraints();
```

Prima si aggiunge l'oggetto `ImageCanvas` al pannello. Poiché si suppone che il controllo `ScrollingPictureWindow` agisca come i controlli nativi dell'AWT, deve essere possibile modificarne le dimensioni. Per specificare che il controllo possa crescere in entrambe le direzioni x e y, si imposta il membro `fill` su `BOTH`:

```
c.fill      = GridBagConstraints.BOTH ;
```

Poiché l'immagine deve occupare tutto lo spazio disponibile senza che venga lasciata alcuna spaziatura, i parametri `weight` vengono impostati su `1.0`:

```
c.weightx   = 1.0;
c.weighty   = 1.0;
```

La disposizione dell'immagine termina richiamando `setConstraints()` per associare l'oggetto `ImageCanvas` all'oggetto `GridBagConstraints`, quindi l'immagine viene aggiunta al layout:

```
gridbag.setConstraints(imageCanvas, c);
add( imageCanvas ) ;
```

Di seguito si dispongono le barre di scorrimento, iniziando da quella verticale, che dovrebbe aumentare o ridurre le dimensioni in senso verticale quando vengono modificate le dimensioni del controllo. Il membro `fill` pertanto viene impostato su `VERTICAL`:

```
c.fill      = GridBagConstraints.VERTICAL ;
```

Se si osserva il layout in termini di righe, si vede che la prima riga contiene due controlli: l'oggetto `ImageCanvas` e la barra di scorrimento verticale. Per indicare che la barra di scorrimento è l'ultimo controllo nella riga, si imposta il membro `gridwidth` su `REMAINDER`.

```
c.gridwidth = GridBagConstraints.REMAINDER ;
```

La disposizione della barra di scorrimento verticale termina associandola all'oggetto `Constraints` e quindi aggiungendola al layout:

```
gridbag.setConstraints(vertBar, c);
add( vertBar ) ;
```

Infine, si dispone la barra di scorrimento orizzontale; poiché deve essere possibile modificarne orizzontalmente le dimensioni, il membro `fill` viene impostato su `HORIZONTAL`:

```
c.fill      = GridBagConstraints.HORIZONTAL ;
```

Il gestore di layout `GridBagLayout` viene utilizzato per evitare che la barra di scorrimento orizzontale riempia l'intera larghezza del controllo, facendo in modo che rimanga della stessa larghezza dell'oggetto `ImageCanvas`. Fortunatamente, la classe `GridBagConstraint` permette di collegare la larghezza di un oggetto a quella di un altro.

Per specificare la larghezza della barra di scorrimento in termini di celle di una griglia si utilizza il membro `gridwidth` della classe `GridBagConstraint`. Impostando questo membro su 1 si fa in modo che la barra di scorrimento orizzontale occupi la stessa larghezza dell'oggetto `ImageCanvas`, vale a dire una cella. È l'oggetto `ImageCanvas` che imposta le dimensioni della cella.

```
c.gridwidth = 1 ;
```

Di seguito si aggiunge la barra di scorrimento orizzontale al layout, dopo averla associata all'oggetto Constraints:

```
gridbag.setConstraints(horzBar, c);
add( horzBar ) ;
```

Per ultimo, si dichiara ScrollingPictureWindow come oggetto AdjustmentListener per entrambe le barre di scorrimento:

```
vertBar.addAdjustmentListener( this ) ;
horzBar.addAdjustmentListener( this ) ;
```

In base a dove viene utilizzato il controllo, può essere possibile modificarne le dimensioni. La modifica delle dimensioni viene gestita ridefinendo il metodo Component.setBounds(), che viene richiamato ogni volta che vengono variate le dimensioni di un controllo. La prima cosa che fa questa funzione è chiamare il metodo setBounds() della superclasse, il quale si occupa dell'effettiva modifica delle dimensioni. Poiché si sta utilizzando un GridBagLayout, il LayoutManager modifica le dimensioni dei singoli componenti:

```
public synchronized void setBounds( int x,
                                    int y,
                                    int width,
                                    int height) {
        super.setBounds( x, y, width, height ) ;
```

Dopo che la superclasse ha modificato le dimensioni, è necessario aggiornare l'immagine e le barre di scorrimento. Innanzitutto si determina se la larghezza del controllo è maggiore della larghezza dell'immagine più la larghezza della barra di scorrimento verticale. In questo caso, si disattiva la barra di scorrimento orizzontale:

```
if ( width > imgWidth + vertBar.getBounds().width ) {
    horzBar.setEnabled( false ) ;
```

Se la larghezza del controllo non è maggiore della barra di scorrimento orizzontale, quest'ultima viene attivata:

```
} else {
    horzBar.setEnabled( true ) ;
```

Successivamente si determina in che modo riposizionare la barra di scorrimento orizzontale. Si inizia ottenendo le dimensioni dell'intero controllo e la larghezza della barra di scorrimento verticale:

```
Rectangle bndRect = getBounds() ;
int barWidth = vertBar.getPreferredSize().width ;
```

Quando si lavora con le barre di scorrimento, è necessario impostare diversi valori:

- ✔ *la posizione della casella;*
- ✔ *i valori minimo e massimo;*
- ✔ *le dimensioni della pagina visualizzabile;*
- ✔ *l'incremento di pagina.*

Ora è possibile calcolare il valore massimo per la barra di scorrimento. Il valore minimo viene sempre impostato su zero, mentre il valore massimo corrisponde alla larghezza dell'immagine meno la larghezza dell'oggetto ImageCanvas. Le dimensioni della pagina e l'incremento di pagina vengono impostati a un decimo delle dimensioni massime:

```
int max = imgWidth - (bndRect.width - barWidth);
block = max/10 ;
```

Prima di impostare i nuovi valori, è necessario determinare come tradurre la vecchia posizione nella nuova scala. Si inizia ottenendo il vecchio valore massimo; se questo è zero, la posizione diventa zero:

```
int oldMax = horzBar.getMaximum() ;
if ( oldMax == 0) {
    imgX = 0 ;
```

Se il vecchio valore massimo non è zero, si calcola la nuova posizione. Innanzitutto si esprime la vecchia posizione come frazione del vecchio massimo, quindi si moltiplica la frazione per il nuovo massimo. Il risultato determina la nuova posizione:

```
} else {
    imgX = (int)(((float)imgX/(float)oldMax) *
                               (float)max) ;
    }
```

L'ultima cosa da fare è impostare i parametri delle barre di scorrimento:

```
horzBar.setValues( imgX, block, 0, max ) ;
horzBar.setBlockIncrement( block ) ;
    }
```

Per la barra verticale si utilizza lo stesso algoritmo.

Aggiunta della classe come ascoltatore

Il controllo della finestra con l'immagine scorrevole è correlato in particolar modo agli eventi delle barre di scorrimento. Tutti gli altri tipi di eventi vengono gestiti all'esterno del programma.

Si inizia implementando l'interfaccia AdjustmentListener e ridefinendo il metodo adjustmentValueChanged(). Essendo l'ascoltatore per entrambe le barre di scorrimento, la classe ScrollingPictureWindow riceve tutti gli eventi di scorrimento. Quando l'utente regola le barre di scorrimento, le variabili imgX e imgY vengono reimpostate e viene richiamato repaint():

```
public void adjustmentValueChanged(AdjustmentEvent e) {
        imgY = vertBar.getValue() ;
        imgX = horzBar.getValue() ;
        imageCanvas.repaint();
    }
```

Unione di tutte le parti

Si ha ora un controllo composito che può diventare un sostituto di altri controlli dell'AWT. Questo controllo gestisce i propri eventi e risponde alle modifiche delle dimensioni. Questa classe viene utilizzata con la classe dell'applet `testPictureWindow` *inclusa nel Listato 21.1, presentata precedentemente in questo capitolo. La classe* `ScrollingPictureWindow` *è inclusa nel Listato 21.2 e si trova anche nel CD-ROM allegato al libro.*

Listato 21.2 *La classe ScrollingPictureWindow.*

```
class ScrollingPictureWindow
                    extends Panel
                    implements AdjustmentListener{

    ImageCanvas imageCanvas ;
    Scrollbar   vertBar ;
    Scrollbar   horzBar ;
    Image       image ;

    int imgWidth ;
    int imgHeight ;
    int imgX ;
    int imgY ;
    int block ;

    public ScrollingPictureWindow ( Image img ) {
        image = img ;
        imgX = 0 ;
        imgY = 0 ;

        while ((imgHeight = image.getHeight(this)) == -1 ) {
        // ciclo fino al caricamento dell'immagine
        }

        while ((imgWidth  = image.getWidth(this)) == -1 ) {
        // ciclo fino al caricamento dell'immagine
        }

        imageCanvas = new ImageCanvas( image ) ;
        vertBar = new Scrollbar( Scrollbar.VERTICAL ) ;
        horzBar = new Scrollbar( Scrollbar.HORIZONTAL ) ;
        GridBagLayout gridbag = new GridBagLayout();
        setLayout( gridbag ) ;

        {
        GridBagConstraints c = new GridBagConstraints();
        c.fill       = GridBagConstraints.BOTH ;
        c.weightx    = 1.0;
        c.weighty    = 1.0;
        gridbag.setConstraints(imageCanvas, c);
        add( imageCanvas ) ;
        }
```

```
        {
        GridBagConstraints c = new GridBagConstraints();
        c.fill       = GridBagConstraints.VERTICAL ;
        c.gridwidth  = GridBagConstraints.REMAINDER ;
        gridbag.setConstraints(vertBar, c);
        add( vertBar ) ;
        }

        {
        GridBagConstraints c = new GridBagConstraints();
        c.fill       = GridBagConstraints.HORIZONTAL ;
        c.gridwidth  = 1 ;
        gridbag.setConstraints(horzBar, c);
        add( horzBar ) ;
        }

        vertBar.addAdjustmentListener( this ) ;
        horzBar.addAdjustmentListener( this ) ;
    }

    public synchronized void setBounds( int x,
                                        int y,
                                        int width,
                                        int height) {

        super.setBounds( x, y, width, height ) ;
        if ( width > imgWidth + vertBar.getBounds().width ) {
            horzBar.setEnabled( false ) ;
        } else {
            horzBar.setEnabled( true ) ;
            Rectangle bndRect = getBounds() ;
            int barWidth = vertBar.getPreferredSize().width ;
            int max = imgWidth - (bndRect.width - barWidth);
            block = max/10 ;
            int oldMax = horzBar.getMaximum() ;
            if ( oldMax == 0) {
                imgX = 0 ;
            } else {
            imgX = (int)(((float)imgX/(float)oldMax) *
                                        (float)max) ;
            }
            horzBar.setValues( imgX, block, 0, max ) ;
            horzBar.setBlockIncrement( block ) ;
        }

        if (height > imgHeight + horzBar.getBounds().height) {
            vertBar.setEnabled( false ) ;
        } else {
            vertBar.setEnabled( true ) ;
            Rectangle bndRect = getBounds() ;
            int barHeight = horzBar.getPreferredSize().height ;
            int max = imgHeight - (bndRect.height - barHeight) ;
            block = max/10 ;
            int oldMax = vertBar.getMaximum() ;
            if ( oldMax == 0) {
                imgY = 0 ;
```

```
            } else {
            imgY = (int)(((float)imgY/(float)oldMax) *
                                                (float)max) ;
            }
            vertBar.setValues( imgY, block, 0, max ) ;
            vertBar.setBlockIncrement( block ) ;
        }
    }

    public void adjustmentValueChanged(AdjustmentEvent e) {
        imgY = vertBar.getValue() ;
        imgX = horzBar.getValue() ;
        imageCanvas.repaint();
    }
};

class ImageCanvas extends Canvas {
    Image canvasImg ;
    public ImageCanvas( Image img ) {
        canvasImg = img ;
    }

    public void paint(Graphics g) {
        if ( getParent().isEnabled() ) {
            g.drawImage( canvasImg,
              -1 * ((ScrollingPictureWindow)getParent()).imgX,
              -1 * ((ScrollingPictureWindow)getParent()).imgY,
              this ) ;
        }
    }
};
```

Un controllo con un'immagine protetto da password

La versatilità dei controlli compositi o estesi risulta evidente quando vengono combinati in un unico applet. L'applet presentato in questo paragrafo è un controllo con un'immagine protetto da password, che combina un controllo `passField` e un controllo `ScrollingPictureWindow`. Nelle Figure 21.3 e 21.4 è mostrato l'applet `testPassWindow`.

Questo applet combina oggetti `passField`, `ScrollingPictureWindow` e `Button`. Poiché le classi `passField` e `ScrollingPictureWindow` sono complete, vengono utilizzate come il controllo `Button` dell'AWT. Poiché gli applet di Java sono componenti riutilizzabili, l'applet stesso risulta essere come un nuovo controllo. Infatti, in Java si creano applet da incorporare nelle pagine Web nello stesso modo in cui si creano controlli da incorporare negli applet.

L'applet `testPassWindow` inizia creando oggetti `passField`, `Button` e `ScrollingPictureWindow`:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
```

Figura 21.3 *L'applet testPassWindow prima di immettere la password.*

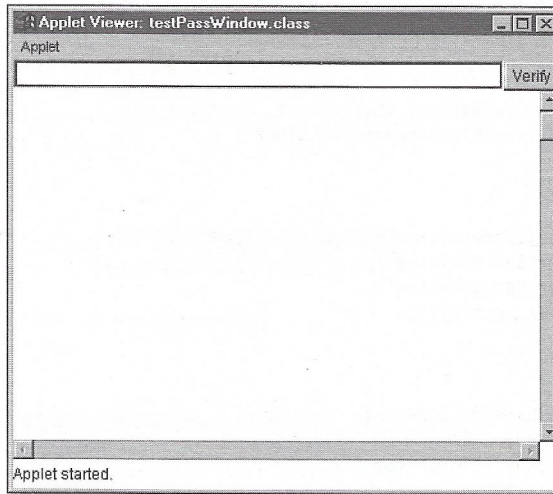
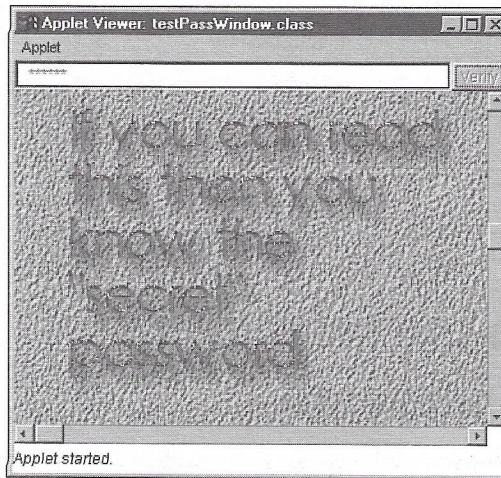



Figura 21.4 *L'applet testPassWindow dopo l'inserimento della password.*

```
public class testPassWindow
                    extends Applet
                    implements ActionListener {
    passField               passwordField ;
    Button                  verifyButton ;
    ScrollingPictureWindow pictureWindow ;
```

Il metodo `init()` crea gli oggetti membro e li inserisce nel controllo utilizzando il gestore di layout `GridBagLayout` (per informazioni su questo gestore di layout e sulle classi `GridBagConstraints` associate, si rimanda al Capitolo 13). Si noti che i controlli `passField` e `ScrollingPictureWindow` vengono utilizzati come se fossero controlli nativi dell'AWT. Infine, il metodo `init()` prenota `testPassWindow` come ascoltatore per il pulsante `Verify`:

```
public void init() {
    GridBagLayout gridbag = new GridBagLayout();
    setLayout( gridbag ) ;
    {
    GridBagConstraints c = new GridBagConstraints();
    passwordField = new passField( 30 ) ;
    c.fill        = GridBagConstraints.BOTH ;
    c.gridx       = 1 ;
    c.gridy       = 1 ;
    c.weightx     = 1.0;
    gridbag.setConstraints(passwordField, c);
    add( passwordField ) ;
    }
    {
    GridBagConstraints c = new GridBagConstraints();
    verifyButton  = new Button( "Verify" ) ;
    c.gridx       = 2 ;
    c.gridy       = 1 ;
    c.gridwidth   = GridBagConstraints.REMAINDER ;
    gridbag.setConstraints(verifyButton, c);
    add( verifyButton ) ;
    }
    {
    GridBagConstraints c = new GridBagConstraints();
    Image img = getImage( getCodeBase(), "picture.gif" ) ;
    pictureWindow = new ScrollingPictureWindow( img ) ;
    c.fill        = GridBagConstraints.BOTH ;
    c.gridx       = 1 ;
    c.gridy       = 2 ;
    c.gridwidth   = 2 ;
    c.weightx     = 1.0 ;
    c.weighty     = 1.0 ;
    gridbag.setConstraints(pictureWindow, c);
    add( pictureWindow ) ;
    }
    pictureWindow.setEnabled( false ) ;
    verifyButton.addActionListener( this ) ;
}
```

Per collegare insieme tutti questi controlli, la classe `testPassWindow` implementa l'interfaccia `ActionListener`. Questa combinazione del controllo con la password e del controllo con la finestra con l'immagine è un buon esempio di incapsulamento in un `Panel`, in modo che diventino un unico grande controllo composito.

La classe `testPassWindow` implementa `actionPerformed()` dell'interfaccia `ActionListener`. Questa funzione controlla semplicemente se la password immessa dall'utente è valida e quindi attiva e disegna l'immagine:

```
public void actionPerformed(ActionEvent e) {
    if ( passwordField.getString().equals( "secret" ) ) {
        pictureWindow.setEnabled( true ) ;
        pictureWindow.imageCanvas.invalidate() ;
        pictureWindow.imageCanvas.repaint() ;
        verifyButton.setEnabled( false ) ;
        passwordField.setEnabled( false ) ;
    }
}
```

Riepilogo

A volte gli applet o le applicazioni necessitano di ulteriori funzionalità per l'interfaccia utente, oltre a quelle fornite dall'AWT. Per estendere l'AWT e creare nuove classi dalle classi di base dell'AWT, è possibile derivare le classi e utilizzare i controlli compositi.

Quando si estende l'AWT, è possibile creare controlli completi che rispondono ai propri eventi. Questi controlli spesso possono essere utilizzati come sostituti dei controlli dell'AWT associati.

La classe `passField` sviluppata in questo capitolo è un esempio di controllo derivato e utilizza le funzionalità di base di un `TextField` dell'AWT, migliorandolo. Il risultato è un controllo che può essere inserito da qualsiasi parte e che può essere utilizzato come un `TextField`.

La classe `ScrollingPictureWindow` creata in questo capitolo è un buon esempio di controllo composito. Questa classe combina i principi delle sottoclassi e dell'incapsulamento: è una sottoclasse di `Panel` e serve per incapsulare il controllo `Canvas` e due barre di scorrimento.

Quando si progetta un applet o un'applicazione in Java, si hanno a disposizione i controlli di base dell'AWT che possono essere combinati e derivati per creare controlli migliori che diventeranno parte della serie personale di strumenti di Java da utilizzare in tutti i successivi progetti di programmazione.